Game Theoretic Analysis of the Slurm Scheduler Model

Wilmer Uruchi Ticona*†,
*Barcelona Supercomputing Center, Barcelona, Spain
†Universitat Politècnica de Catalunya, Barcelona, Spain
E-mail: wilmer.uruchi@bsc.es

Keywords—SLURM, Scheduling, High Performance Computing, Game Theory.

I. Extended Abstract

In the context of High Performance Computing, scheduling is a necessary tool to ensure that there exists acceptable quality of service for the many users of the processing power available. The scheduling process can vary from a simple *First Comes First Served* model to a wide variety of more complex implementations that tend to satisfy specific requirements from each group of users. Slurm is an open source, fault-tolerant, and highly scalable cluster management system for large and small Linux clusters [1]. MareNostrum 4, a High Performance Computer, implements it to manage the execution of jobs send to it by a variety of users [2]. Previous work has been done from an algorithmic approach that attempts at directly reduce queuing times among other costs [3][4]. We consider that there is utility at looking at the problem also from a Game Theoretic perspective to define clearly the mechanics involved in the system, and also those that define the influx of tasks that the scheduler manages. We model the Slurm scheduling mechanism using Game Theoretic concepts, tools, and reasonable simplifications in an attempt to formally characterize and study it. We identify variables that play a significant role in the scheduling process and also experiment with changes in the model that could make users behave in a way that would improve overall quality of service. We recognize that the complexity of the models might derive in difficulty to theoretically analyze them, so we make use of usage data derived from real usage from BSC-CNS users to measure performance. The real usage data is extracted from Autosubmit [5], a workflow manager developed at the Earth Science Department at BSC-CNS. This is a convenient choice, given that we also attempt to measure the influence of an external agent (e.g. a workflow manager) could have in the overall quality of service if it imposes restrictions, and the nature of these restrictions.

A. Slurm Overview

The Slurm scheduling mechanism has two main components: Priority and Scheduler. **Priority**: A value calculated based on data from the user, and the jobs that the user sends to the High Performance Computer (HPC). The calculation tries to give higher priority to those users that have less usage. **Scheduler**: Once the jobs have been received and their Priority calculated, there are certain rules in Slurm that determine when a job is sent for execution. As a result, the job with the highest priority is not always executed first, but the order is altered so resource usage is optimized.

Users Hierarchy: The users are organized in a hierarchical tree structure, specifically in a Rooted Plane Tree [6], where on top of it we have a main **root** account. Then, the leafs are users and the internal nodes are the accounts to which they are associated. In the Slurm documentation we encounter many references to the term **account**, we consider it equivalent to the term **group**.

Priority Calculation: There are four factors involved in the calculation of this value: **Age**, a value from 0.0 to 1.0. The longer a job sits in the queue and is eligible to run, the bigger this value gets. **Size**, a value from 0.0 to 1.0 determined by the number of nodes a job requests, the more nodes a job requests the bigger this value gets. **Fair-share**, a value from 0.0 to 1.0 determined by [7]. **QoS**, a value from 0.0 to 1.0 determined by the priority given to different QoS defined in the Slurm implementation.

Scheduling Mechanism: The **Priority** value effectively produces an execution sequence; however, this ordering can result in sub optimal resource allocation. For example: *Consider a large (in the size of nodes required) job with high priority that is waiting to be scheduled, this job will take 25% of the nodes in the HPC and it has a planned (supplied by the user) running time of 10 hours; furthermore, it is in the front of the queue. After this job, we have a number of smaller jobs requiring a number of nodes from 1% to 2% of the total nodes in the HPC, and expected times lower than 1 hour.* Working under this standard configuration, the scheduler is going to wait for enough resources to be available and then schedule the large job for execution, and this is not optimal because resources will be idle. To avoid this scenario (to some extent), there exists the **backfill** mode. In this mode, the scheduler will start lower priority jobs if that does not delay the start of higher priority jobs.

B. Data Overview

Slurm receives jobs, these jobs come from experiments on which the users are working on. A typical experiment can be modeled as a directed acyclic graph (DAG). It starts with jobs that retrieve or set information, or they might compile software that will be used in the later stages of the experiment. Then, there are some heavy computation tasks that usually consist on simulations that implement parallel processes and subsequently require many nodes and long running time. As we mentioned,

these experiments can be modeled as a **DAG** $G = (V, E)$ where we have $V = \{1, ..., n\}$ tasks and $V^{`}$ as the list of tasks sorted in topological order with sizes $w_1, w_2, ..., w_n$ measured in the number of **HPC** nodes they require, and $t_1, t_2, ..., t_n$ as the planned time in minutes they will need to complete. Typically, a number of vertex at the beginning and end of $V^{`}$ will require less computation resources than the rest in average. We will avoid using the word "node" to name the vertex in a graph to avoid confusion with **HPC** nodes.

C. Game Theoretic Perspective

We begin by assuming that the backfill scheduling mechanism subject of study is working as a **BF_MOD** scheduler as defined in [3]. Under this assumption, resources are reserved when a job reaches the top of the queue and is about to be scheduled and the next jobs in the queue can be used for backfilling, but if in the next scheduling event a job with higher priority takes position at the top of the queue, the previous reservation is discarded and a new reservation is executed for the newly arrived top priority job. On the other hand we have **BF_UNMOD** where the top job does not change even if a higher priority job arrives, this is also defined in [3]. We assume our scheduler under **BF_MOD** configuration for further discussion.

There are M nodes in the HPC, there are U users that handle experiments represented as **DAGs** $G_u = (V_u, E_u)$ that result in a topological order of jobs $V_u^{`} = \{1, 2, ..., n\}$ for each experiment $G_u \in G$ where $G$ is the set of all experiments from users $U$. Users will send job $v_i \in V_u^{`}$ when $\alpha_i = \{v_j | (v_j, v_i) \in E_u \ \wedge status(v_j) \neq 5\} = \emptyset$, where $status()$ is a function that returns 5 if the input job v has status COMPLETED, meaning that all preceding jobs of v_i must have been completed successfully. Then, we define the set of jobs sent to the scheduler by user u at a given iteration as $\nu_u = \{v_i | v_i \in V_u^{`} \wedge \alpha_i = \emptyset\}$.

$$N = \bigcup_{u \in U} \nu_u$$

Think of the scheduler as an agent Λ that receives N jobs, each job $n \in N$ has attributes w_n for its weight or size, t_n as the planned time (supplied by the user) in minutes that the job will take to complete, a_n for the time it has been waiting for execution, and p_n for its **Priority** calculated using the previously mentioned attributes. Agent Λ uses the **Priority** p_n as the main ordering principle to define a list $P_k : N \longrightarrow \{1, ..., n\}$ of execution order. We now proceed to give a glimpse of the analysis of games modeled using these definitions.

Single Attempt Game: We have N jobs ordered in a priority list P_k that defines the order in the queue of jobs to be scheduled for execution. A **strategy profile** s is an assignment of d jobs to $|M|$ nodes by an agent Λ. In a single backfill scheduling attempt, agent Λ takes the first d jobs in the ordering P that minimize $\varepsilon = |M| - \sum_{i=1}^{P_d} w_i$. Then, the utility u of agent Λ is $u(s) = |M| - \varepsilon$. Then, we proceed to analyze the Nash Equilibrium of this setup as a foundation stone for the analysis of more complex games.

Multiple Attempt Backfill Game: Agent Λ actions under a sequence of backfill iterations. Clearly, it is under this scenario that the backfill mechanism comes into play.

Submit to Scheduler Game: Instead of analyzing a single agent Λ, we have players U submit jobs to an independent scheduler. We define a utility function for each player based on some of the variables that the user can modify when sending a job.

Multiple Submit to Scheduler Game: Players U submit jobs to an independent scheduler under a sequence of iterations.

D. Experimental Environment

The workflow manager Autosubmit [5] saves the submit, start, and finish time of each job it manages. We will use this data to model synthetic data and test the behavior that, according to the Game Theoretic analysis, shows promising results. Slurm implements a node simulator that can be used for this purpose. We will also try to simulate different traffic situations in the arrival of jobs in order to test worst case scenarios, if possible.

E. Conclusion

In this study, we try to use the Game Theoretic approach to analyze an existing system in order to detect opportunities for the implementation of configurations or tools that potentially result in an improvement of quality of service. However, this study might also reveal that there design choices or configurations that make any kind of collaboration or control not convenient.

II. Acknowledgment

This work is being developed in close collaboration with the Computational Earth Science team at BSC-CNS.

References

[1] "Slurm documentation." [Online]. Available: https://slurm.schedmd.com/archive/slurm-17.11.7/overview.html

[2] "Marenostrum4 user's guide." [Online]. Available: https://www.bsc.es/user-support/mn4.php

[3] Baraglia R., Capannini G., Pasquali M., Puppin D., Ricci L., Techiouba A.D, "Backfilling strategies for scheduling streams of jobs on computational farms," *Making Grids Work*, 2008.

[4] Sergei Leonenkov, Sergey Zhumatiy, "Introducing new backfill-based scheduler for slurm resource manager," *Procedia Computer Science*, vol. 66, pp. 661–669, 2015.

[5] E. S. D. at BSC-CNS, "Autosubmit." [Online]. Available: https://autosubmit.readthedocs.io/en/latest/

[6] S. G. W. Edward A. Bender, *Foundations of Combinatorics with Applications*, 2006.

[7] "Fair tree fairshare algorithm." [Online]. Available: https://slurm.schedmd.com/archive/slurm-17.11.7/fair_tree.html

Wilmer Uruchi Ticona received his BSc degree in Systems Engineering from Private University of Tacna (UPT), Tacna in 2011. Then, he started working in a variety of large scale software development projects for his alma mater and other private companies. In 2018 he decided to pursue further education taking the Masters in Innovation and Research in Informatics (Advanced Computing specialization) program at Universitat Politècnica de Catalunya (UPC), Spain. Since 2019, he has been with the Computational Earth Science group of Barcelona Supercomputing Center (BSC), Spain.